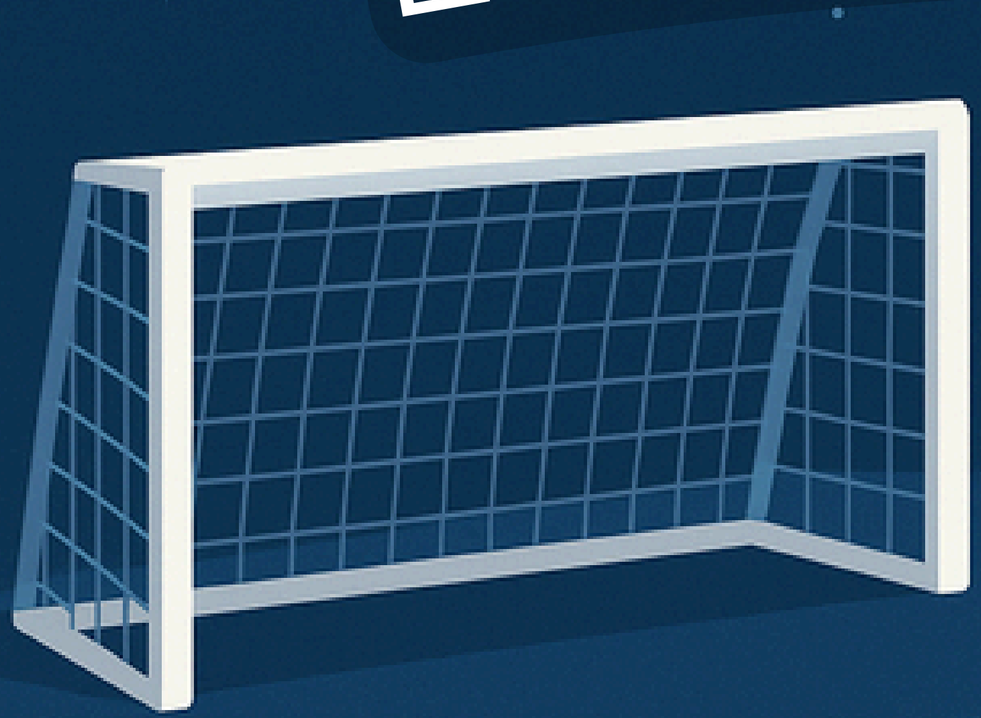
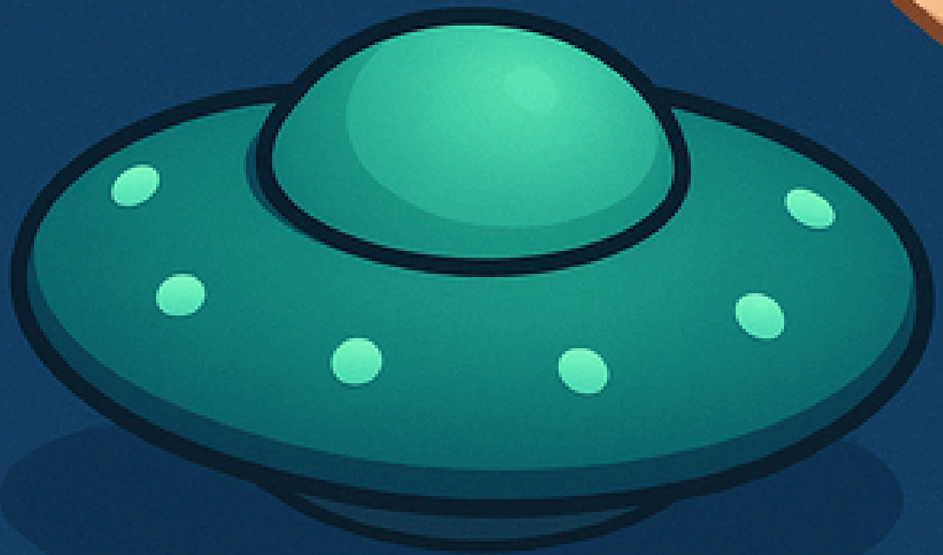
E.T. Telefono Calcio

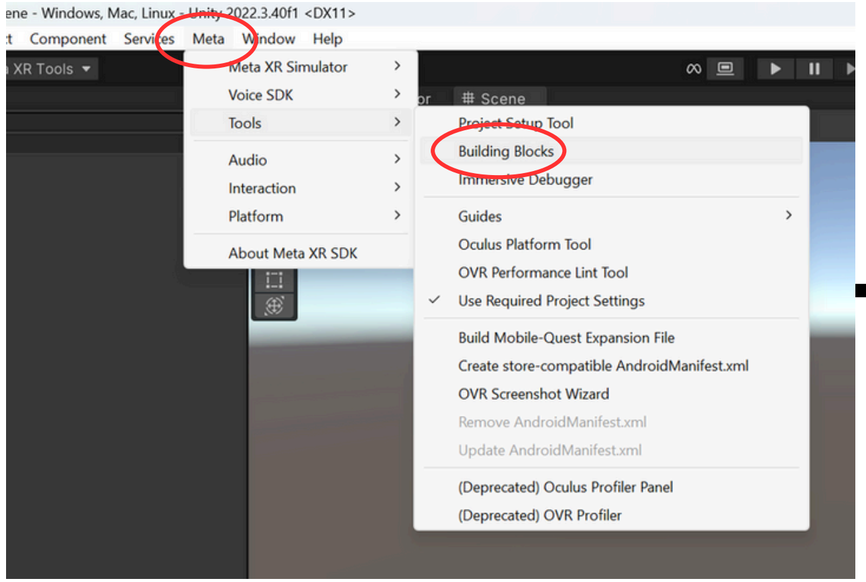


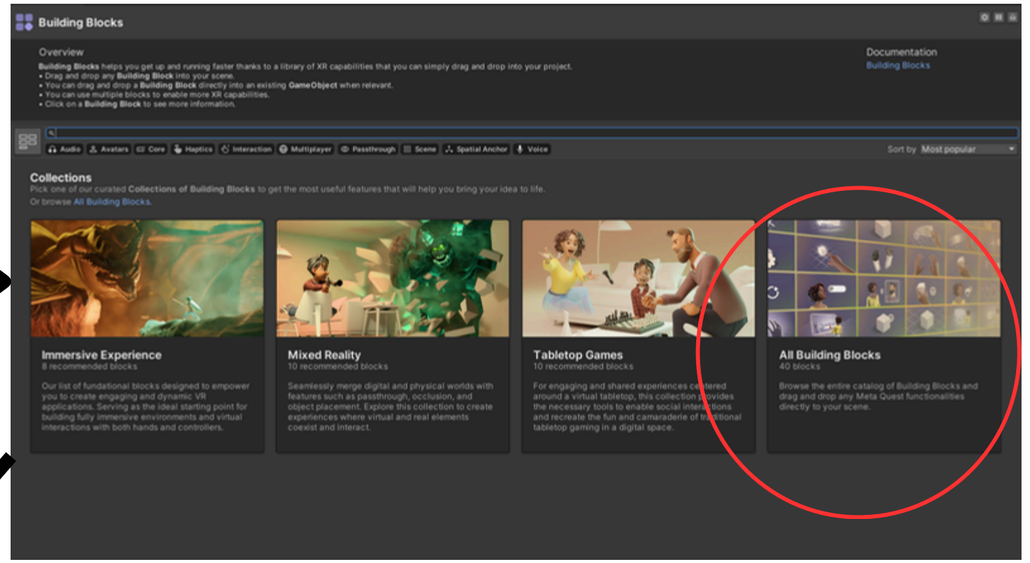


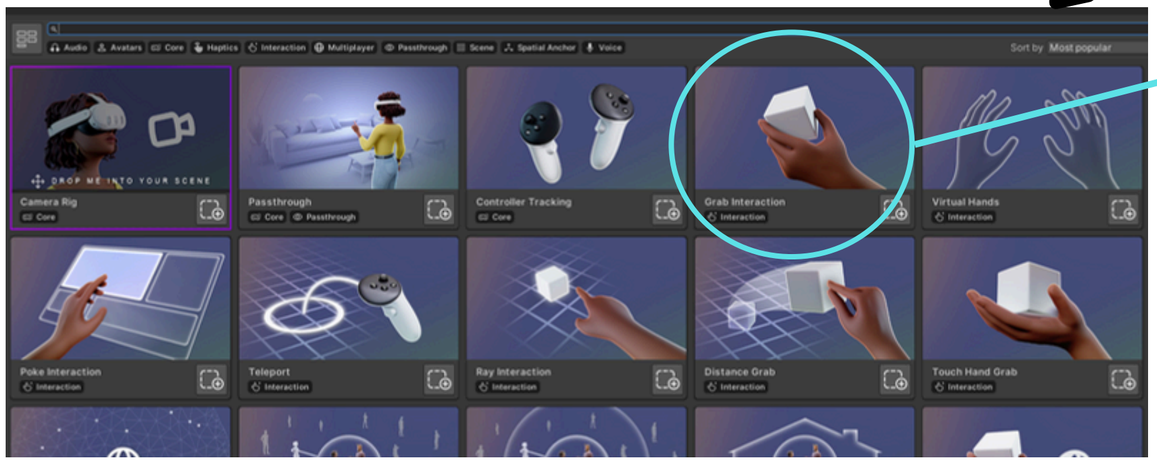
Unity AR Tutorial

Meta Building Blocks

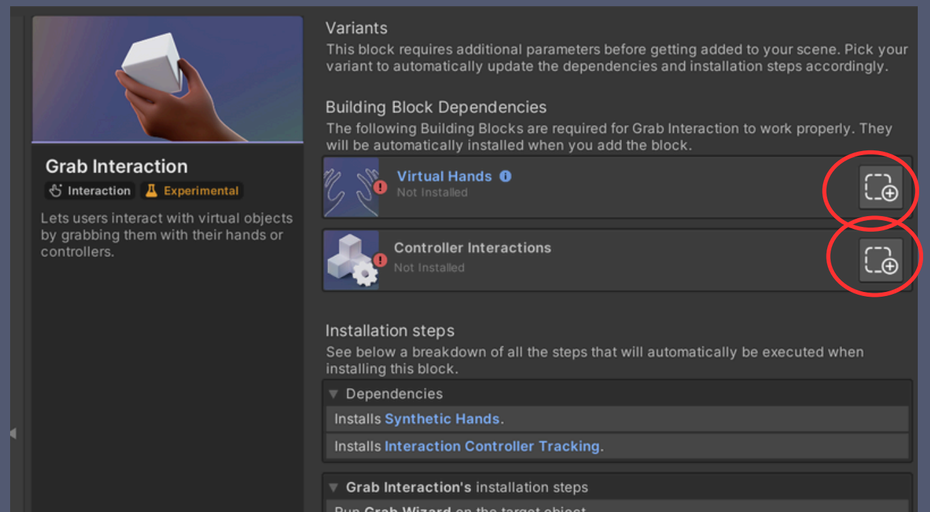
Enter the wizard





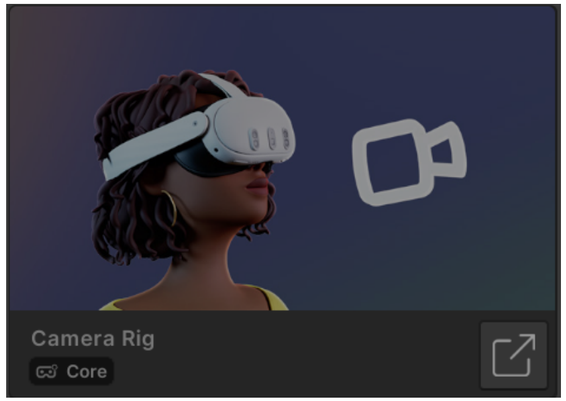


The addition of one block may necessitate the addition of another preparatory block



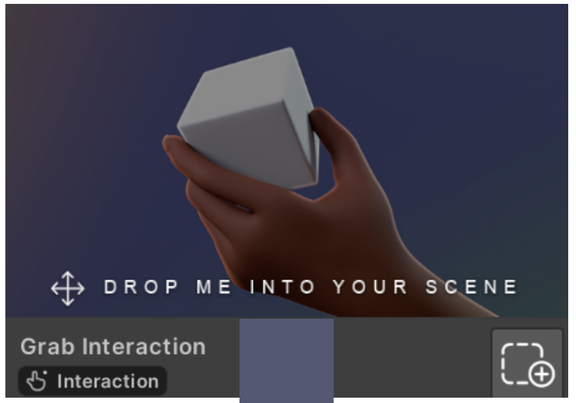
Aggiungete:

Spawn of the avatar with cameras & colliders

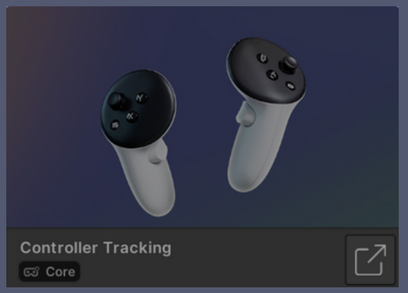


It allows us to see the real environment

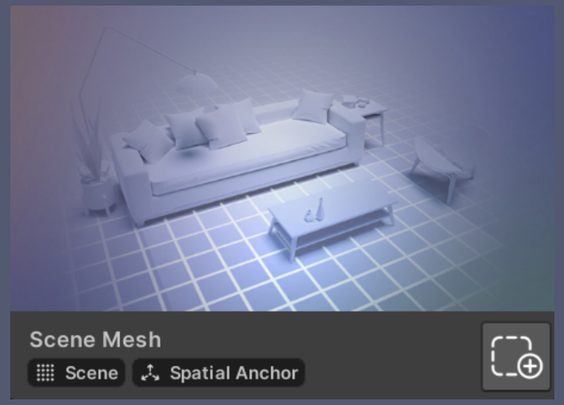
It will allow us to grab the soccer goal and resize it and recognize the user's hand shape



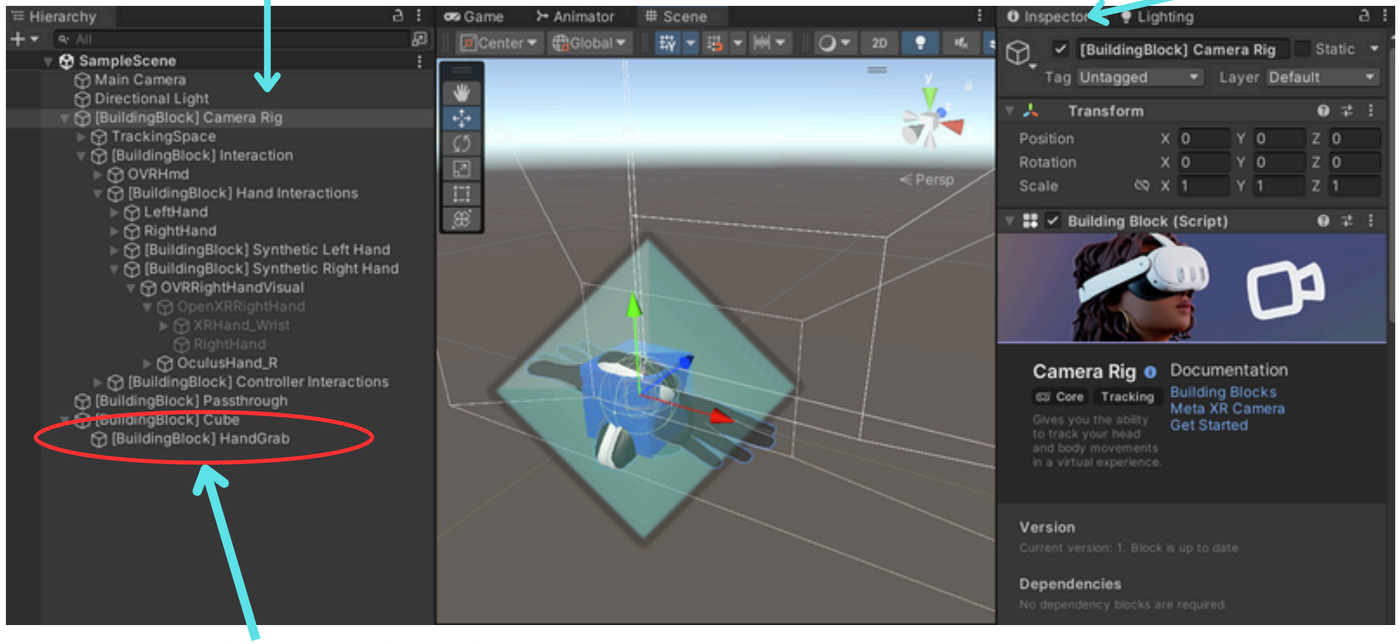
Propedeutici



Opzionale, se volete esplorare

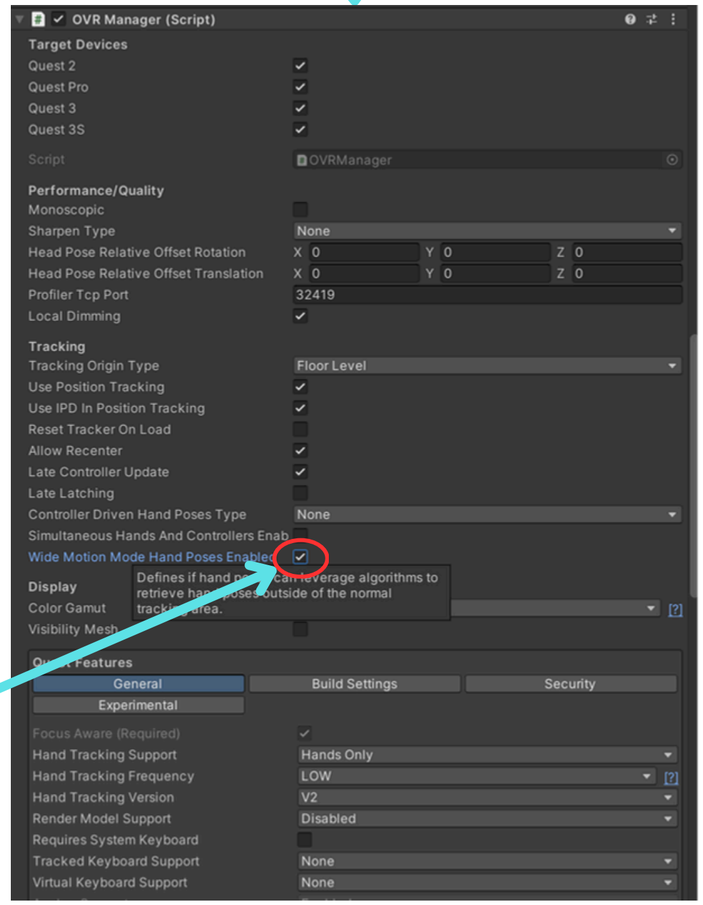


The addition of the blocks will have produced a hierarchy like this

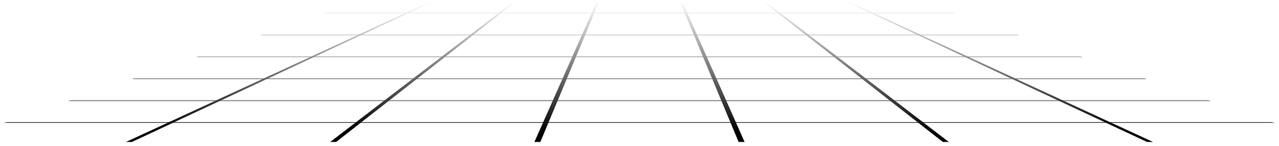


Let's go and remove the cubes (example) that were created right click / Delete

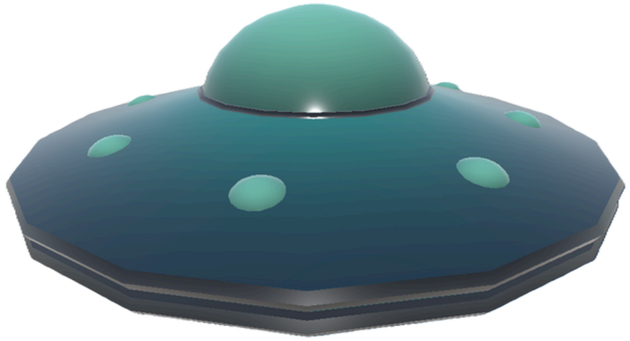
we select in the hierarchy the gameObject “Camera Rig” and modify its components in the Inspector panel

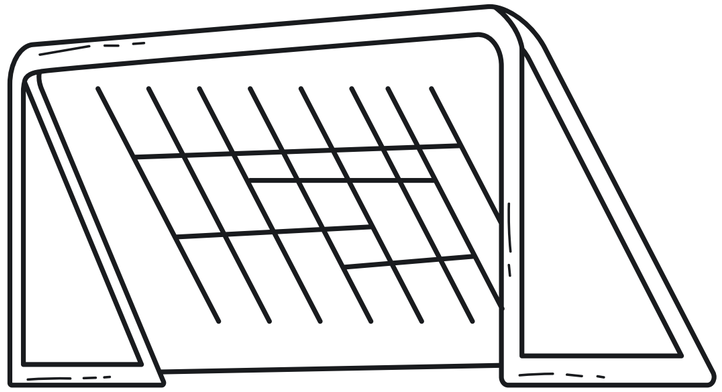


We activate the algorithms for an estimate of the position of the hands outside the FOV



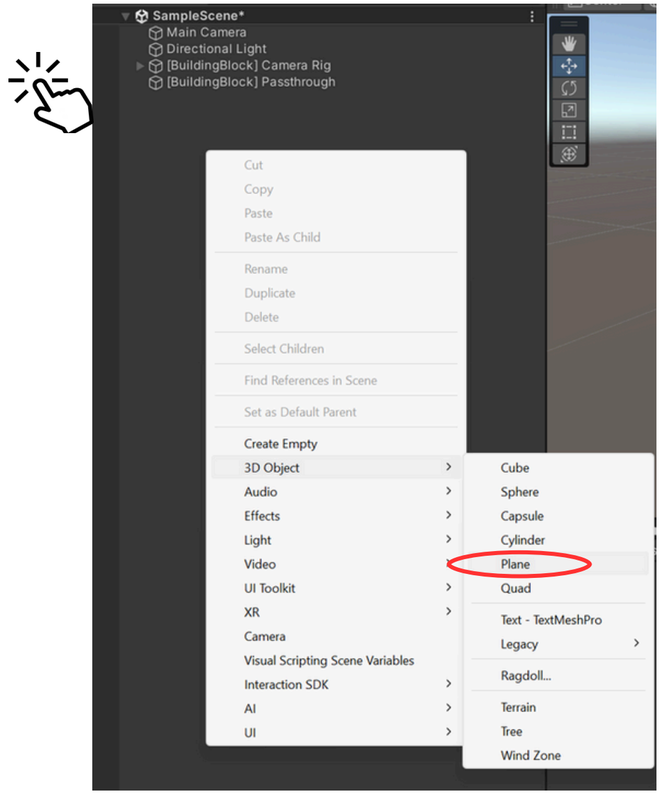
The scene





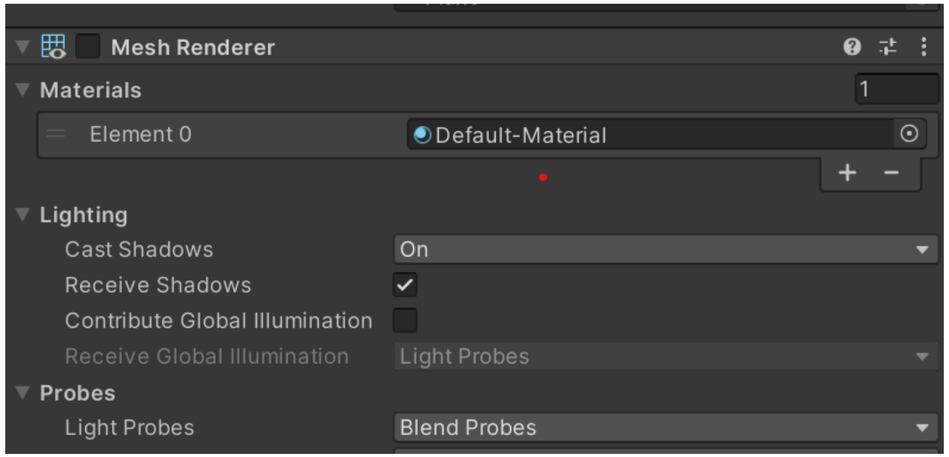
Floor

We add in the scene hierarchy (at level 0) a plane.



Let's change in the inspector its Transform: position (0,0,0) because the Camera Rig considers the floor y = 0, and size (100, 100, 100) so as not to drop the ball

We disable the floor renderer, because since we are in AR we only want to see the real floor.



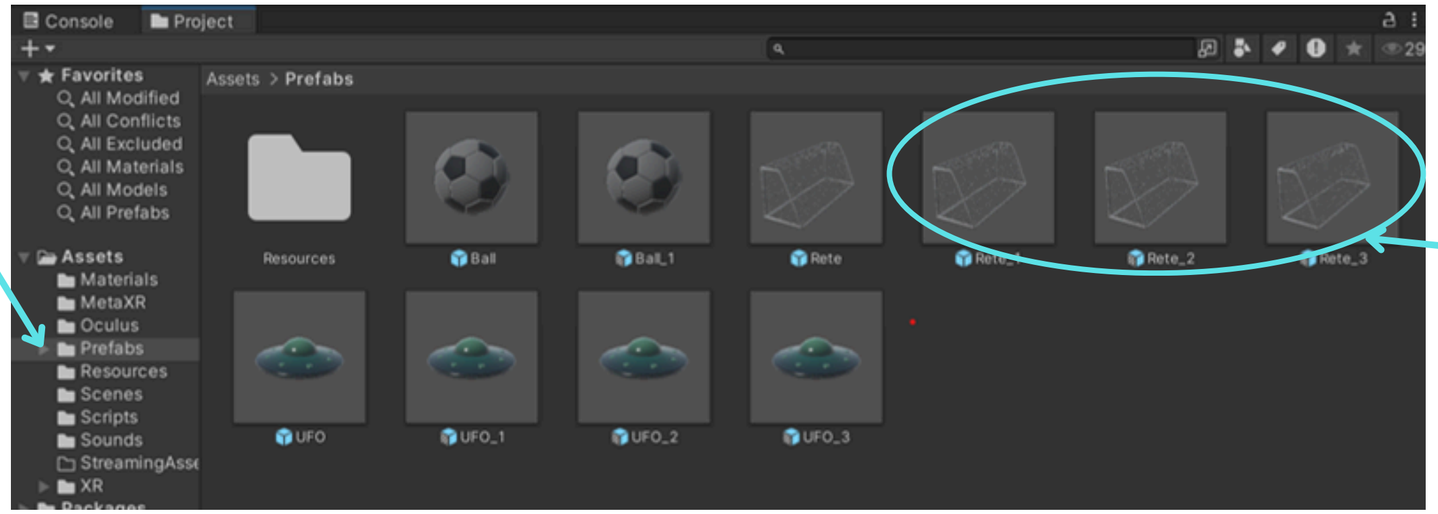
Prefabs

Once you create from a gamobject, you can save it in the project files (scene independent) and afterwards you can clone it anywhere without redoing the work from scratch

with the benefit..

Changing the original Prefab is like changing the design: all copies follow the new design

You can find them at Assets/Prefabs accessible from the “Project” panel.



The variants of prefabs

You can make ‘variants’ that maintain the link to the original but have unique ADDITIVE features

variants of prefab “Rete”

We use only the first variant and modify it incrementally

If you don't keep up, you can import the appropriate variant

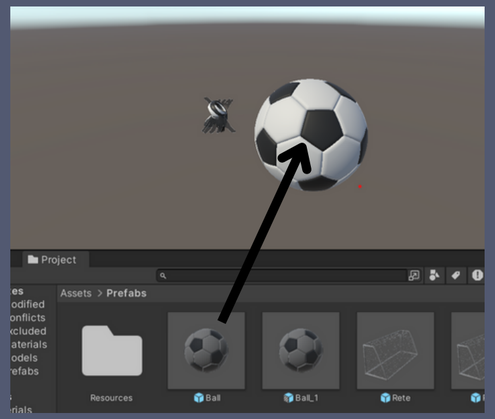
Drag it in scene!

To spawn it in a similar position (same y) as the Camera Rig, before dragging it into the scene select in “Hierarchy” the Camera Rig

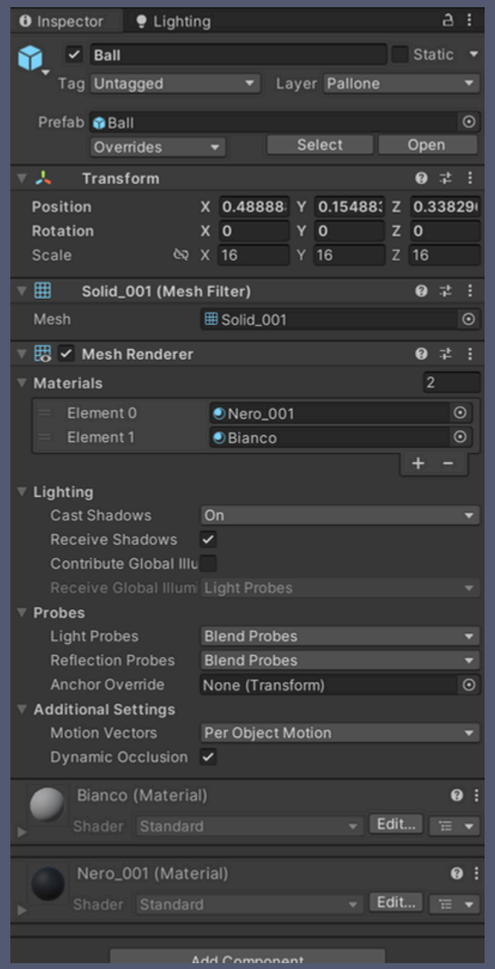
The Ball

A passive object that doesn't need our scripts

But... **Unity PHYSICS**



After selecting “Camera Rig” Drag prefab “Ball” on stage.



You will see appear in the “Inspector” the list of Components attached to Ball:

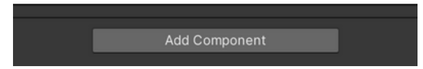
Mesh Filter

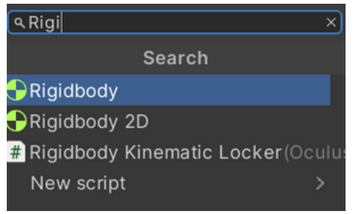
+

Mesh Renderer

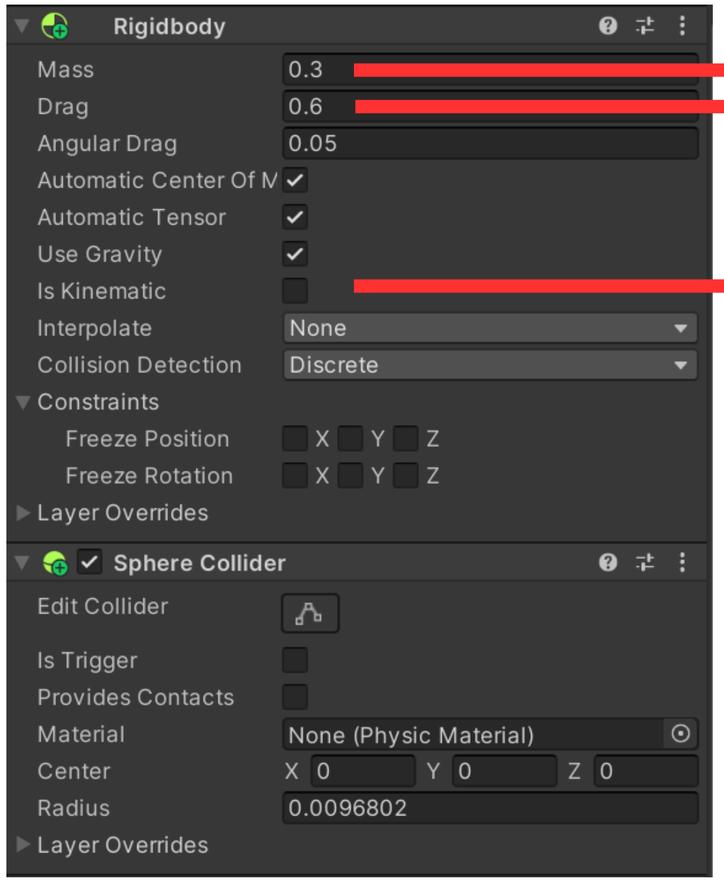
The ball can be seen but does not physically exist

Let's attach the componets:





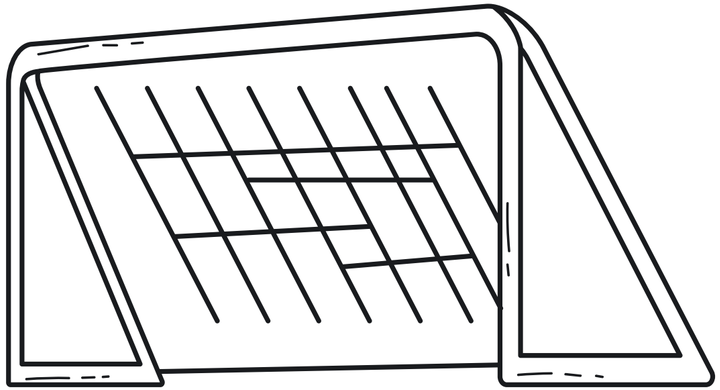
Rigidbody + Sphere Collider



Mass of 0.3 units
Drag of 0.6

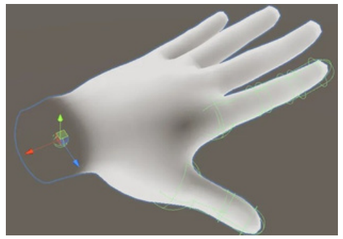
(**Drag** expresses how much friction affects body motion: 0 → no friction, 1 → ball rolls indefinitely)

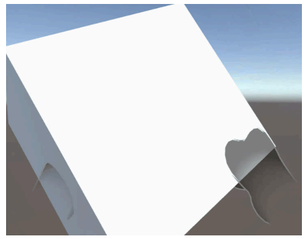
Is Kinematic == false otherwise there would be no elastic shock



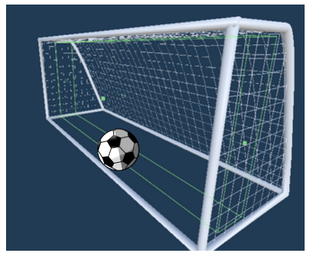
After selecting “Camera Rig” Drag the “**Rete**” prefab in scene

The net (Rete)

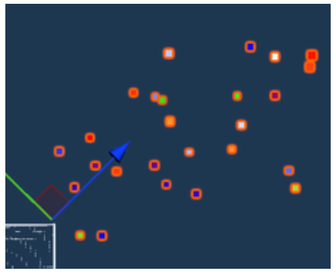




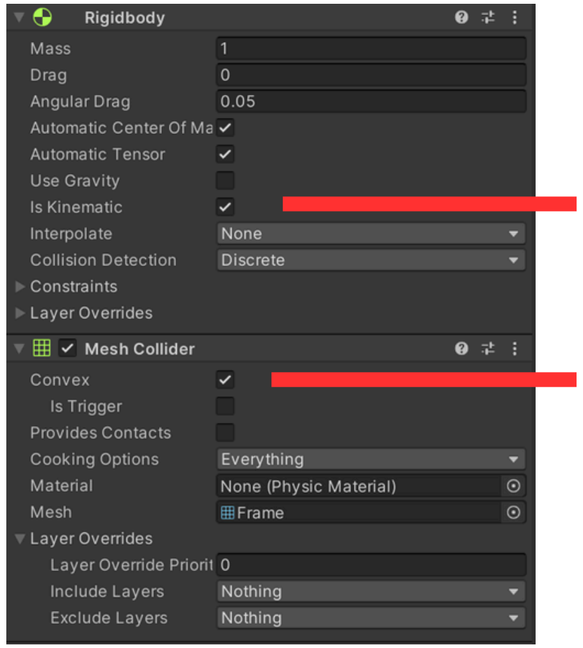
Grabbable Object & Scalable Object with Hand Gestures



Response from Custom script to activation of a Trigger Collider



Particles Effects
(To *cheer the Goal*)



We add to the gameobject Mesh (first level):
Rigidbody + Mesh Collider

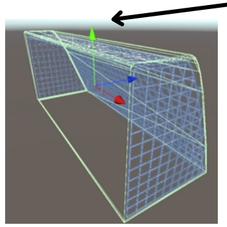
isKinematic = true because we don't want the object to react to collisions with the ball

We have limitations imposed by the GRABBING feature:

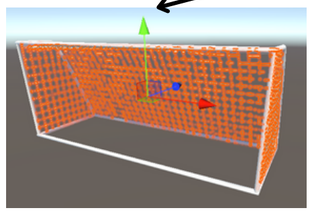
- Must have a **Rigidbody**
- Must have a **Collider** indicating possible graspable points
- if the collider is a Mesh type then it must have **Convex = true**

BUT the ball cannot enter the soccer goal because the current collider is a convex (solid) block...

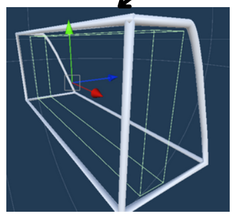
Let's use 3 different Colliders: **2 phisical and 1 Trigger**



A referment collider for the Grab component convex.
But for the ball, it does not have to exist



A non-convex physical collider to stop the ball in the right surfaces



A collider inside the door of trigger type notifies us if the ball goes through it

isKinematic

Unity `Rigidbody.isKinematic` Behavior Summary

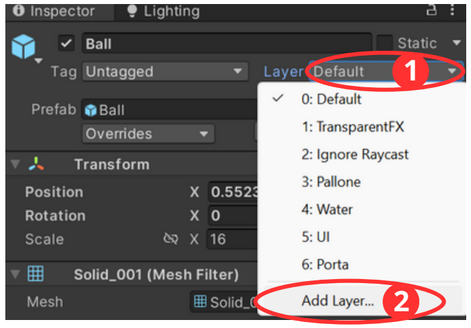
Behavior/Feature	`isKinematic = false` (Dynamic Body)	`isKinematic = true` (Kinematic Body)	No Rigidbody (Static Collider)
Affected by gravity	✅ Yes	❌ No	❌ No
Affected by physics forces	✅ Yes (`AddForce`, `AddTorque`)	❌ No	❌ No
Can move via transform/MovePosition	❌ Not recommended (causes tunneling)	✅ Yes	✅ Yes
Moves other objects on contact	✅ Yes (via collision response)	✅ Only if moved manually	❌ No
Can be pushed by other objects	✅ Yes	❌ No	❌ No
Participates in collision events	✅ Yes (`OnCollisionEnter`, etc.)	⚠️ Only when moved and colliding with **dynamic** rigidbodies	❌ No
Supports trigger events	✅ Yes	✅ Yes (if collider is `isTrigger`)	✅ Yes (if collider is `isTrigger`)
Generates physics contacts	✅ Yes	⚠️ Limited (only when moved manually)	❌ No
Supports interpolation/extrapolation	✅ Yes	✅ Yes	❌ No
Responds to collision layers/masks	✅ Yes	✅ Yes	✅ Yes
Works with joints (e.g., HingeJoint)	✅ Yes	⚠️ Limited use, acts as an anchor	❌ No
Can be part of ragdoll physics	✅ Yes	⚠️ Only temporarily (e.g., for pose control)	❌ No
Used for animation-controlled objects	❌ No	✅ Yes (recommended way)	✅ Yes (but no physics)

We want everything to collide with everything except:
Ball ⇔ convex net

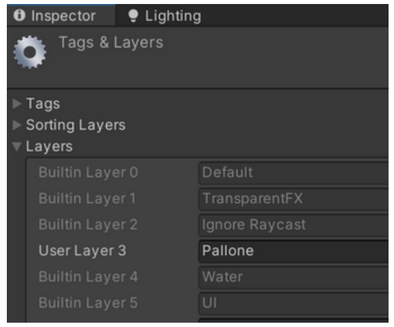
Layers

They help us treat a collection of gameobjects differently.

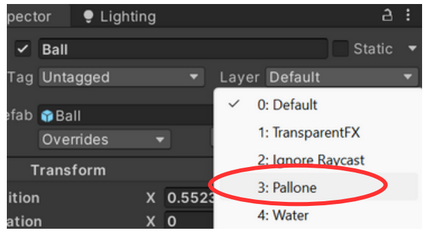
1 We modify the layer of the Ball



We click on the “Layer” drop-down menu then “Add Layer”... in the “Ball” inspector.



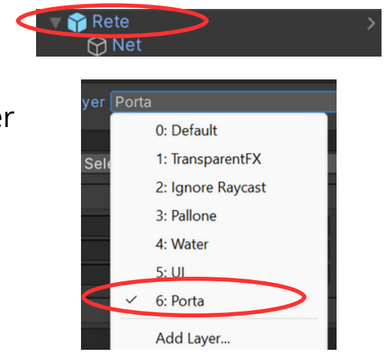
Let's add an item named “**Pallone**” to the list.



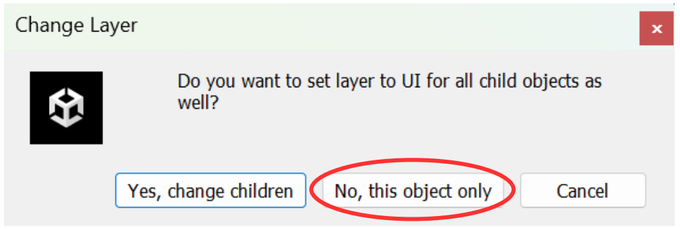
Let's go back to inspect “Ball” and set the new Layer

2 We edit the layer of “Rete”

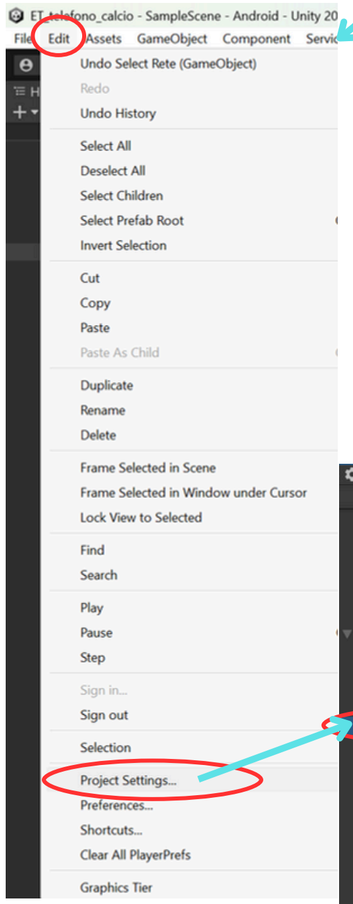
We create and assign a new Layer called “**Porta**” to the gameobject “Rete”



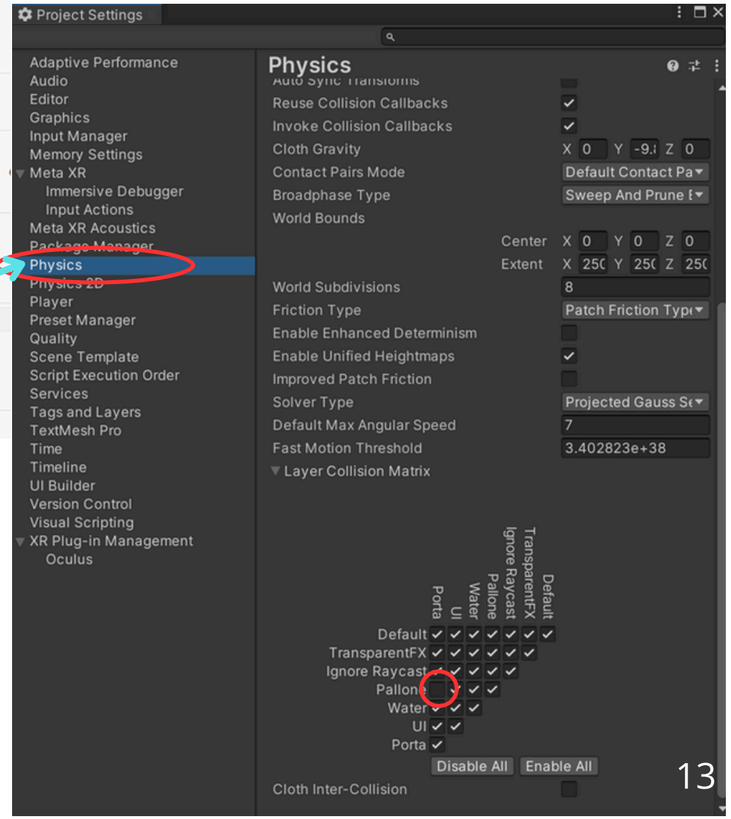
And when it asks us whether to change it for the children as well, we choose **“no”**

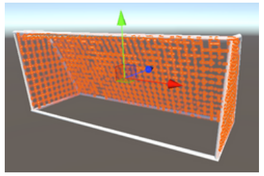


3 Let’s move to Project Settings.. / Physics

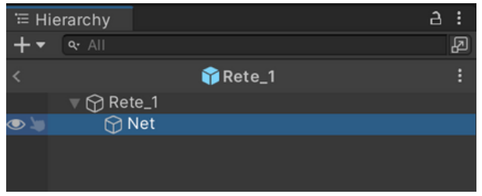


Let's scroll down the panel and look for a grid of checkboxes called “**Layer Collision Matrix**”.
Let's uncheck the one between Goal and Ball.

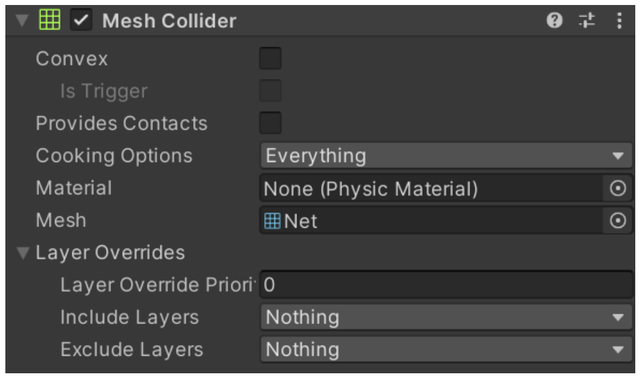




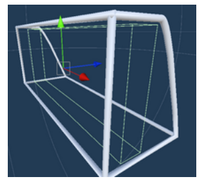
A non-convex physical collider to stop the ball in the right surfaces



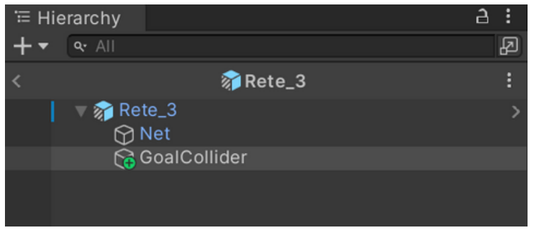
We highlight the child “**Net**”



We attach a **Mesh Collider** component, and we set:
Convex = false & Is Trigger = false

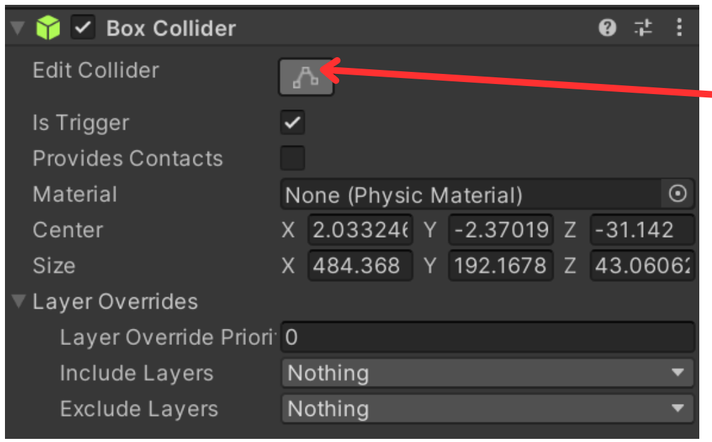


A collider inside the door of trigger type notifies us if the ball goes through it

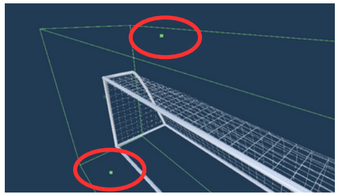


Let's create a new child inside the gameobject “Rete”:
Right-click above the gameobject and click on “Create Empty”

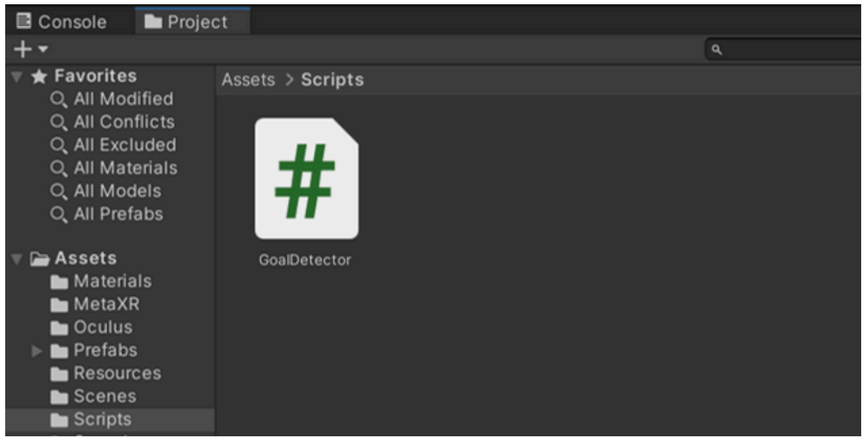
We add a **Box Collider** to the newly created child and set:
Is Trigger = true.
Imagine that if the ball touches the box then the goal has occurred



Edit the Boxcollider parallelepiped by clicking this button.
In the scene use the active (green) points on the boxcollider to resize it.



Let's create a script that allows us to recognize a goal and respond accordingly



Let’s go to the directory Assets/Scripts in the panel “Project”

Then.. right click in the empty space, click on “**Create**” → “**C# Script**”

And we call it “**GoalDetector**”

Derived class from MonoBehaviour

classes that are derived from MonoBehaviour can be attached to GameObjects as components.

\+ Your custom MonoBehaviours

UnityEngine automatically recognizes some functions defined in MonoBehaviour scripts, such as **Start**(), **Update**(), and **OnCollisionEnter**().

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(Collider))]
public class GoalDetector : MonoBehaviour
{
    private int palloneLayer;

    private void Awake()
    {
        palloneLayer = LayerMask.NameToLayer("Pallone");
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.layer == palloneLayer)
        {
            Goal(other.gameObject);
        }
    }

    private void Goal(GameObject pallone)
    {
        // TODO Cheering feedback
        StartCoroutine(MoveBallAfterDelay(pallone.transform, 2.5f));
    }

    private IEnumerator MoveBallAfterDelay(Transform ballTransform, float delay)
    {
        yield return new WaitForSeconds(delay);
        ballTransform.position += -transform.forward * 1f;
    }
}
```

RequireComponent(typeof(Collider)) asserts that the gameobject (***objSoggetto***) has a component of type Collider (***myCollider***).

Awake() of ***objSoggetto*** is invoked at the beginning of the scene along with all the Awake of each component on each gameobject

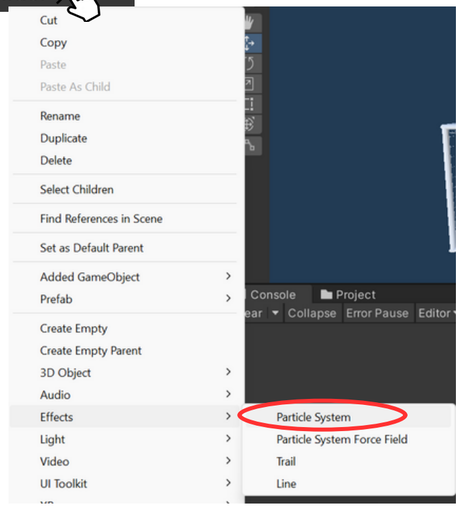
OnTriggerEnter(Collider *other*) is invoked when a collider ***other*** intersect ***myCollider***.

StartCoroutine(IEnumerator routine**)** it is a function made available by Unity to perform “almost” asynchronous routines.
yield return allows us to pause the execution of that code until a condition occurs (without stopping all other scripts), in this case we wait 2.5 seconds.

1 We add particles to celebrate each goal

Right click on the child created i previously “GoalCollider”

Effects → ParticleSystem

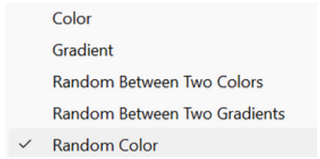


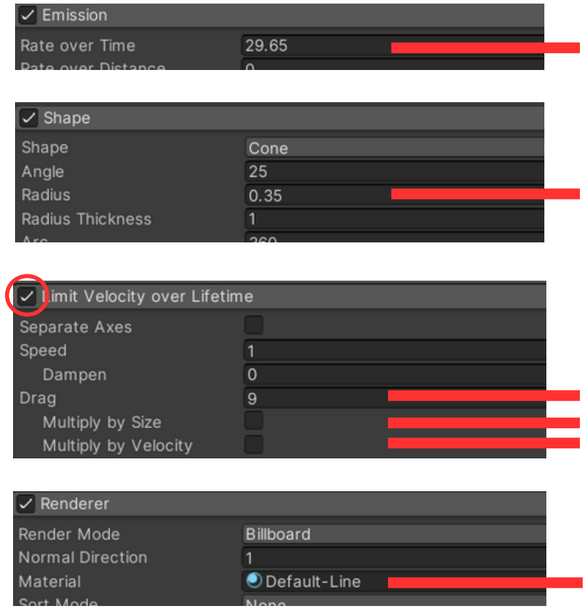
2 Let's edit in the editor the component ParticleSystem

Start speed, **Start Lifetime**, **gravity modifier**, **Drag** together define the motion and for how long it is followed by each particle.

Duration = 2: we want the cheering to be just 2 seconds

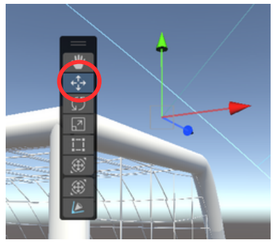
Start Color, click on the drop-down menu on the right and select “Random Color” and add color dots



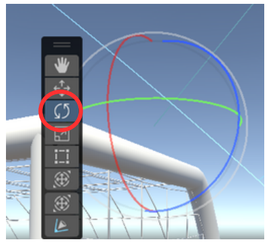


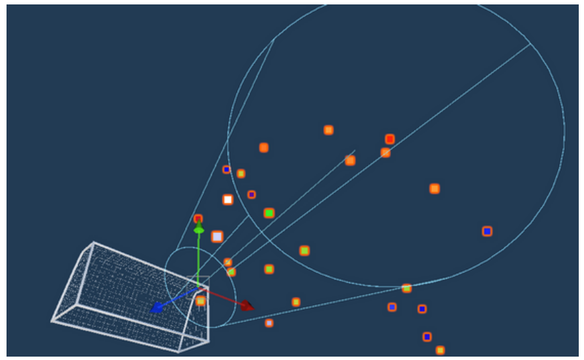
3 we move the cone from where the particles are generated to our liking

Using the translation control...



...And the rotation one





4

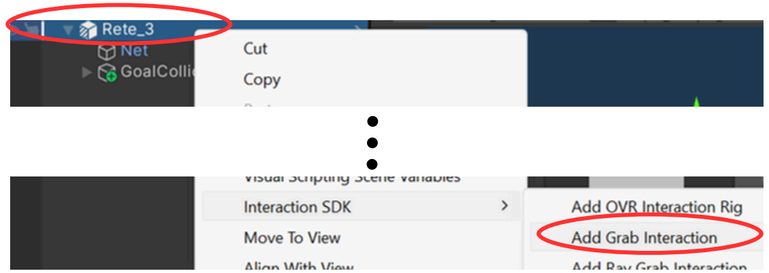
We clone the new gameobject “Particle System” if we want to have two particle sources.

5 Let's edit the script “GoalDetector” to activate the particles at Goal

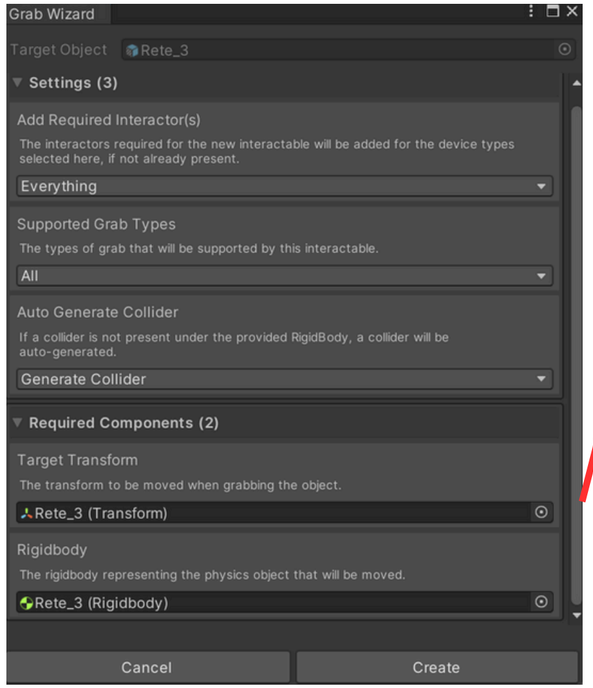
```
....
    private void Goal(GameObject pallone)
    {
        ParticleSystem[] particles =
            GetComponentsInChildren<ParticleSystem>();
        foreach (var ps in particles)
        {
            ps.Play();
        }
        StartCoroutine(
            MoveBallAfterDelay(pallone.transform, 2.5f));
    }
....
```

We make the goal net graspable and resizable with hand gestures

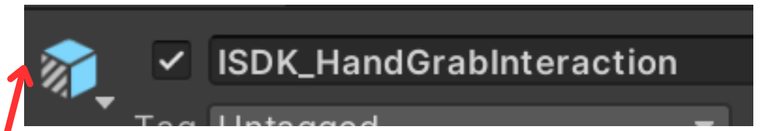
Right click on the root of the gameobject “Rete,”
then: Interaction SDK → Add Grab Interaction



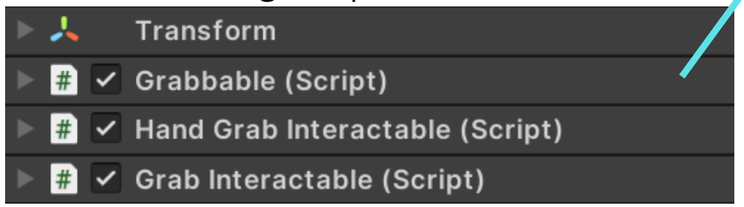
A wizard will open that will generate a new child for our gameobject Rete.
Rete must have the components:
Collider e **Rigidbody**



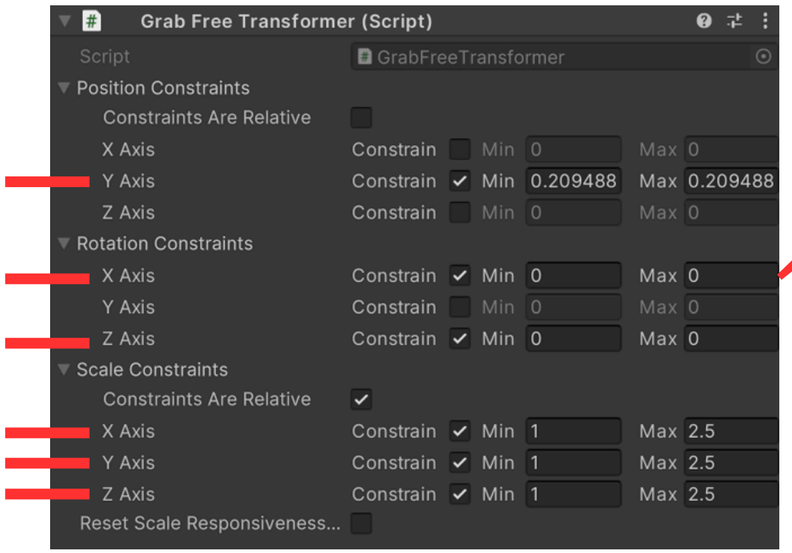
OUTPUT:



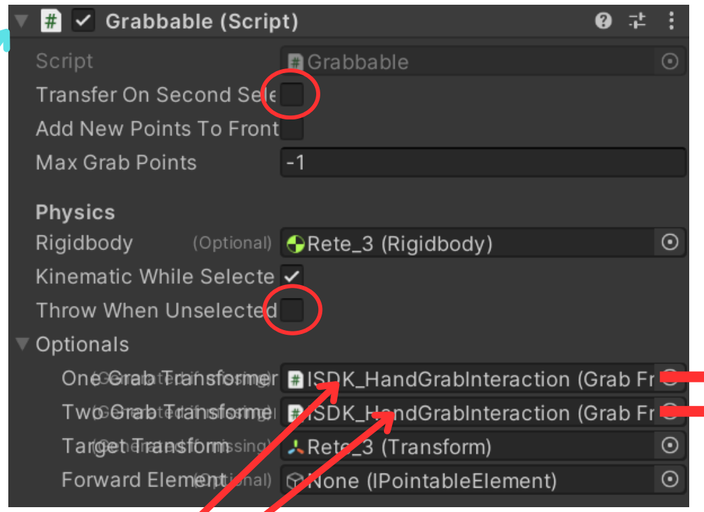
With the following components:



Then, we attach the component
“**Grab Free Transformer**”:



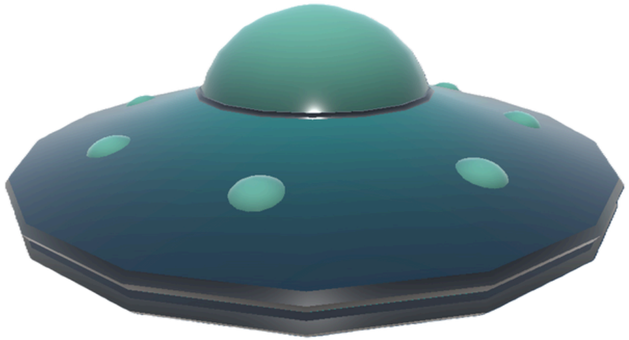
We edit the autogenerated component
“Grabbable”



Dragging the previously added “Grab Free Transfomer” component into the fields of “**Grabbable**”:

- One Grab Transformer
- Two Grab Transformer

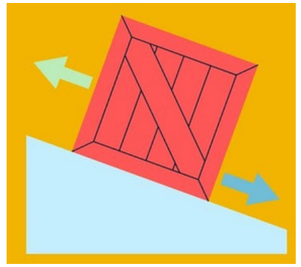
We indicate to the meta SDK that moving will be done with one hand and scaling with two



After selecting “Camera Rig” Drag prefab “**UFO**” to the scene.

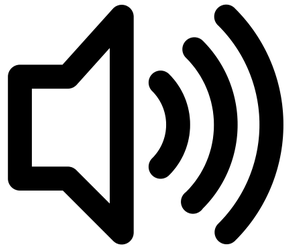
L’ UFO

Cheap Gesture Recognition



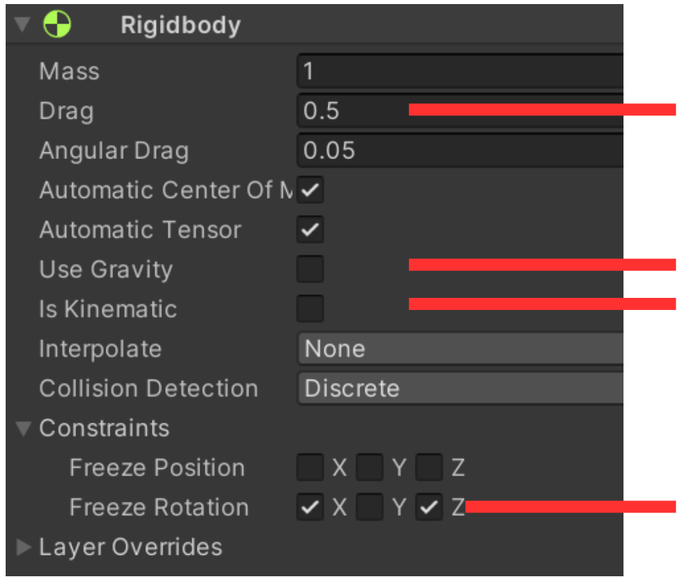
Navigation with forces

Colliders with physical materials



Spatial Audio Sources

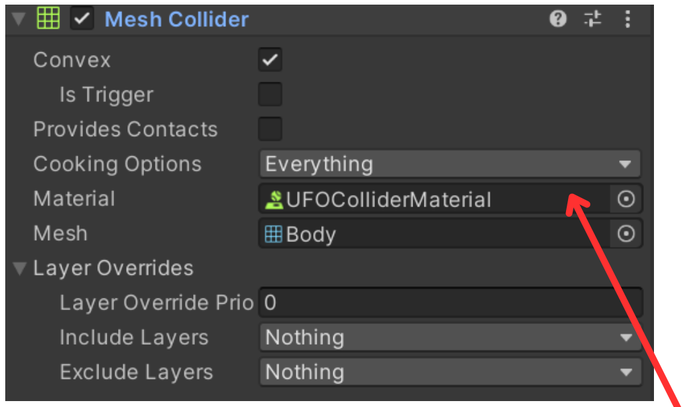
We want the **Rigidbiody** of non-kinetic type like that of the ball (it will have to respond realistically to 'impact).



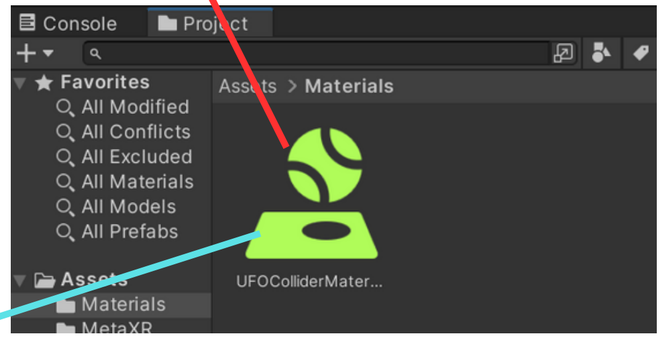
Since the rigidbody is **nonkinetic** it makes sense to use constraints to block unintended rotations of the object produced by shocks.

We don`t want rotations in the x and y axes.

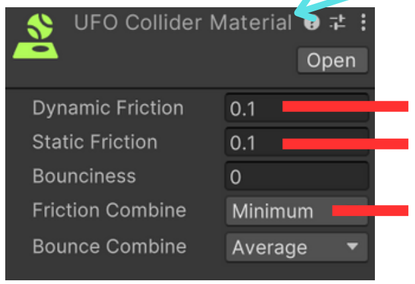
We add a **Mesh Collider** and refine the collision behavior with the **physical material**



Let's go to Assets/Materials then, on the empty space: right click → Create → Physical Materal



Once the material is selected, we can configure it with the following values:

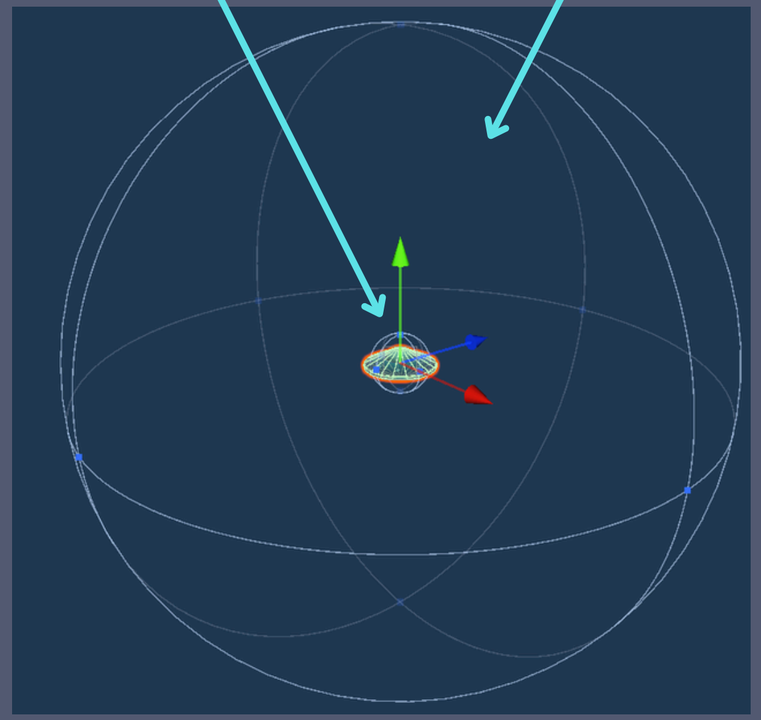


We reduce the friction almost completely to steer the UFO with more agility even when it is forced by the pavement.

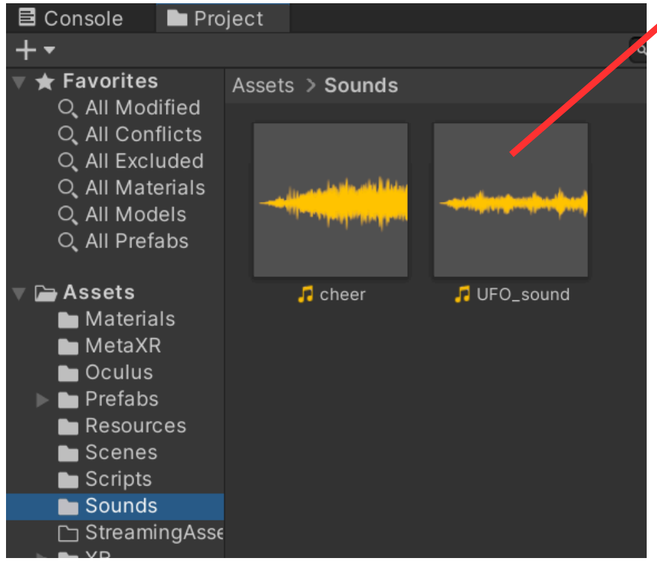
We want a spatial audio source for the spacecraft to increase immersion

Inner sphere: from where start the logaritm. function

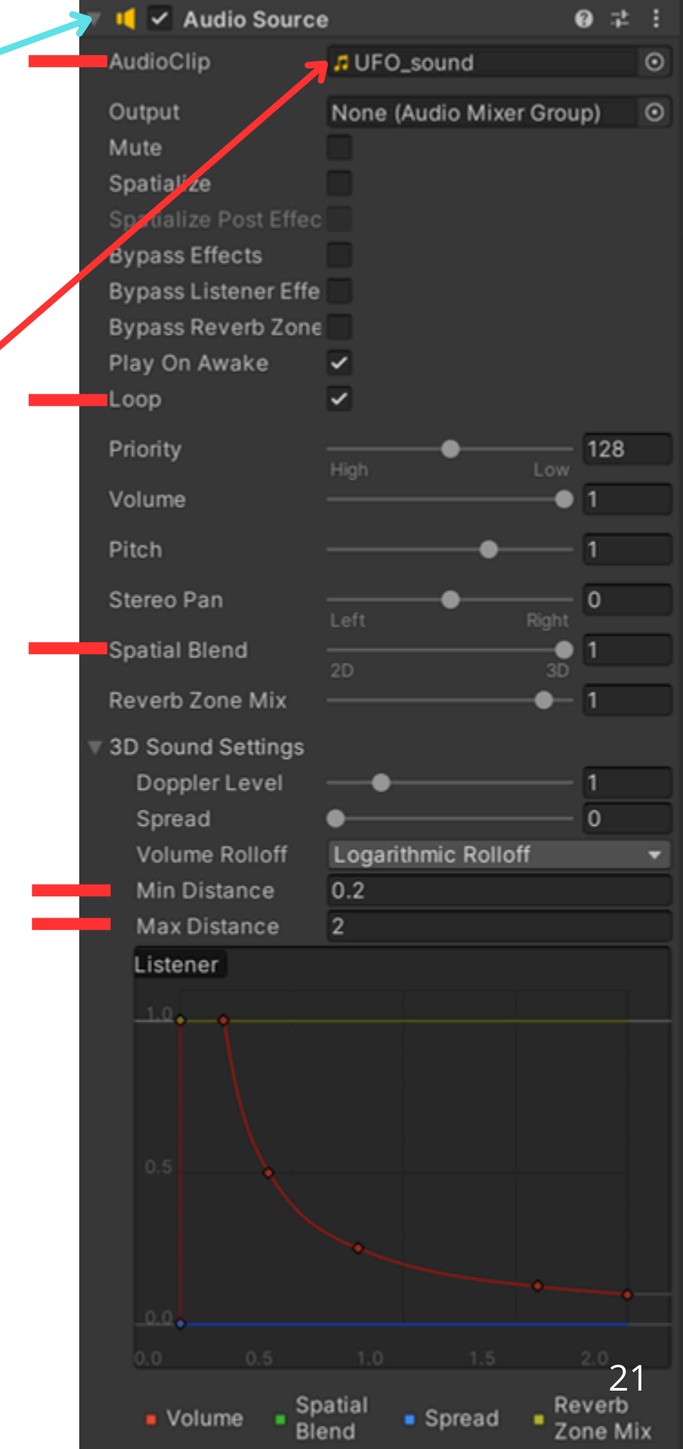
Outer sphere. where the audibility ends



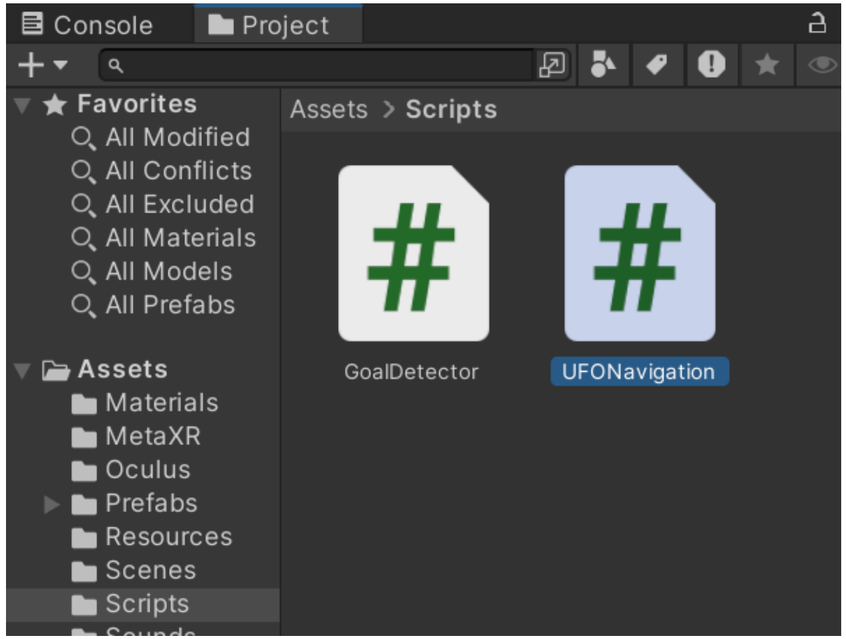
We must then attach a component of type **AudioSource** to the gameobject “UFO”



We find the **AudioClip** to assign to the AudioSource at:
Assets/Sounds/UFO_sound



Let's now focus on the UFO movement



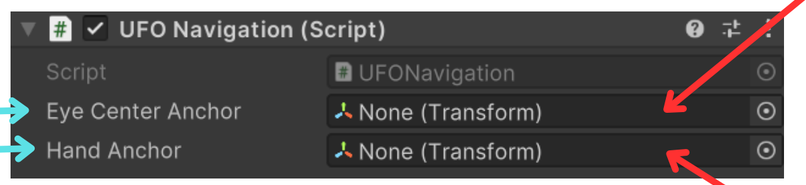
We make a new script to attach to the UFO gameobject called:
"**UFONavigation**"

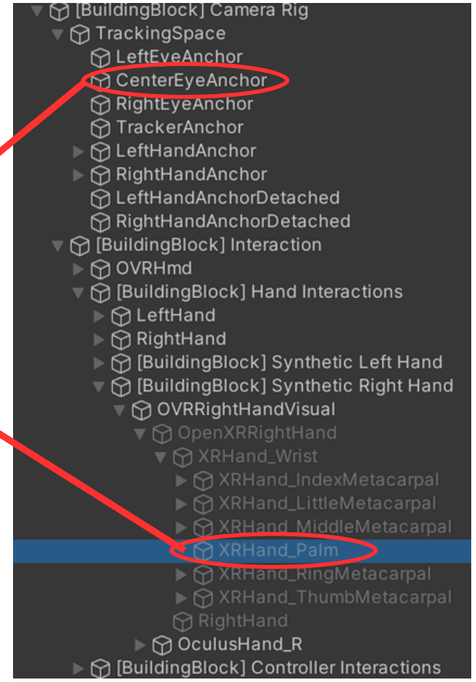
We want to achieve in sucessive steps the following behaviors:

- A **constant rotation** of the UFO around the y-axis by 1 turn every 5 sec
- The UFO follows a target point (Vector3). We need to **apply forces** to minimize its distance from the target.
- The target is controlled by the user: the target is located in the **line** passing through the **POV and the user`s hand**.
- the **depth of the target** in space is given by the degree of **openness** of the **user`s hand**.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class UFONavigation : MonoBehaviour
{
    public Transform eyeCenterAnchor;
    public Transform handAnchor;

    private float angularSpeed = Mathf.PI * 0.4f;
    private Rigidbody myRigidBody;
    private float responsivnessToPosError = 15f;
    private float maxForce = 5f;

    void Start()
    {
        myRigidBody = gameObject.GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        Vector3 targetLocal = (handAnchor.position -
            eyeCenterAnchor.position).normalized;

        Vector3 positionError = targetLocal - transform.position;
        Vector3 controlForce = Vector3.ClampMagnitude(
            responsivnessToPosError * positionError, maxForce);

        myRigidBody.AddForce(controlForce, ForceMode.Force);

        myRigidBody.angularVelocity = new Vector3(0f, angularSpeed, 0f);
    }
}
```

Once the script is attached to the gameobject "UFO" we modify the component in the inspector





We drag from the hierarchy the gameobjects:

- CenterEyAnchor → in the field "eyeCenterAnchor"
- XRHand_Palm → in the field "handAnchor"

gameObject.GetComponent<Rigidbody>() looks in the components attached to ***objSoggetto*** (the UFO) for the first one of type Rigidbody.

FixedUpdate() is a method called (for each obj) as the beginning of PHYSICS-related cyclic operations.
USEFUL LINK: https://docs.unity3d.com/es/2019.4/Manual/ExecutionOrder.html

We want to move the body with forces so we use: **AddForce**(forceVector, ForceMode.Force).
But we don't care that the rotation is physically handled so instead of using **AddTorque** we go directly to set a constant angular momentum with "myRigidBody.**angularVelocity**"

To recognize the degree of openness of the hand, we want to measure the distance of the middle finger from the palm.

```
public Transform handAnchor;
public Transform palmo;
public Transform puntaDito;

public float openHandCoeff = 0.13f;
public float closedHandCoeff = 0f;

public float maxDistanceUFO = 4f;

private float currentOpennessCoeff;
```

we exchange handAnchor with **palm** e **middlefinger point**

At each cilclo we find **currentOpennessCoeff** ∈ [0, 1]
to arrive at this normalization we need to know the minimum distance of the middle finger from the palm (**closedHandCoeff**) and the maximum (**openHandCoeff**)

at each cycle we will impose the distance of the target point from the proportion:

0(currentOpennessCoeff**)** → **0(**dist**)**
1(currentOpennessCoeff**)** → **maxDistanceUFO(**dist**)**

```
private void Update()
{
    Vector3 delta = puntaDito.position - palmo.position;

    float forwardDistance = Vector3.Dot(delta, palmo.forward.normalized);
    Debug.Log(forwardDistance);
    forwardDistance = Mathf.Clamp(forwardDistance, closedHandCoeff, openHandCoeff)

    currentOpennessCoeff = (forwardDistance - closedHandCoeff ) / openHandCoeff;
}
```

we add to the script the **Update()** function, invoked by Unity each cycle (related to scripts, not physics).

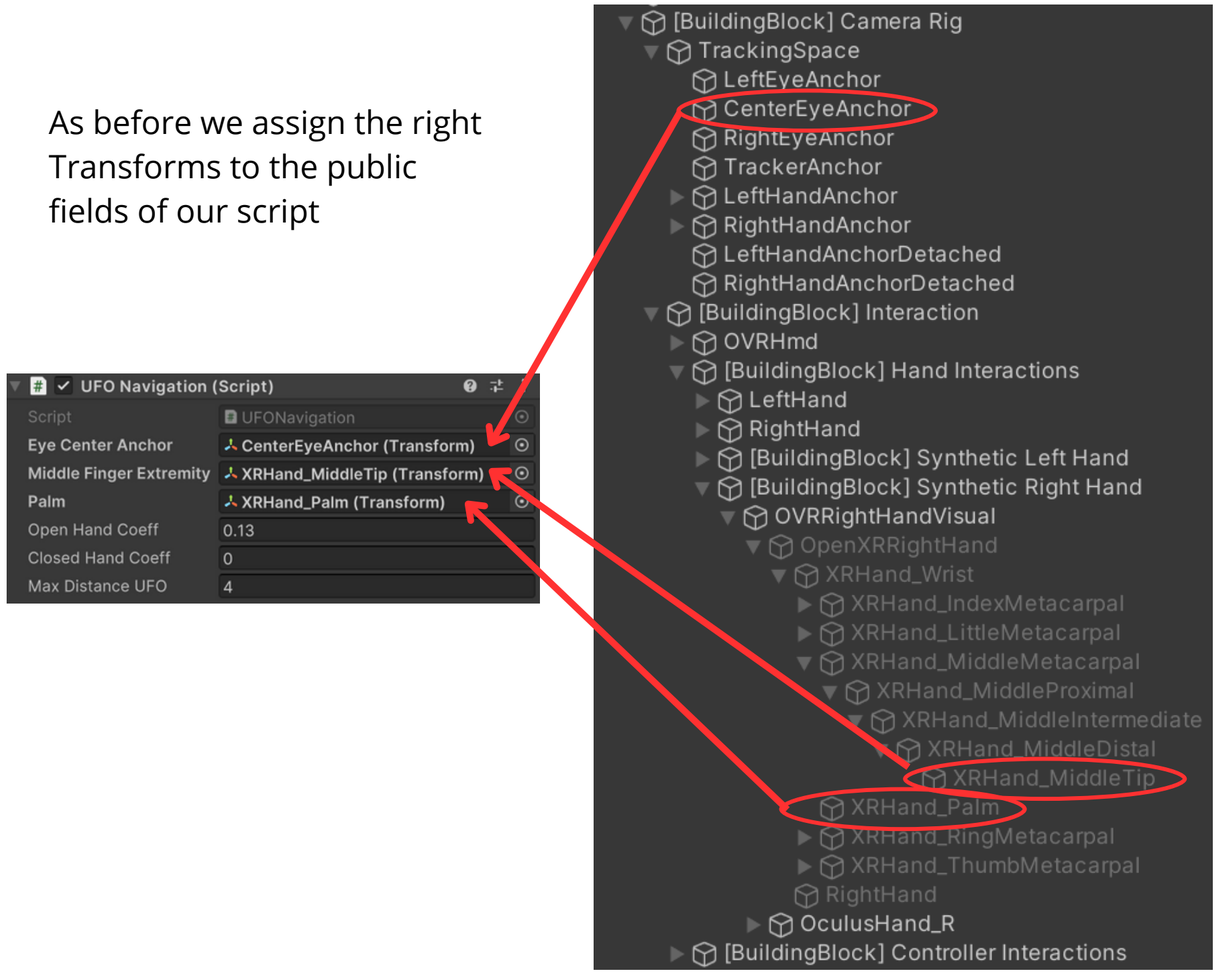
forwardDistance is the mid-palm distance projected onto the line indicating the direction of the hand.

Debug.Log(forwardDistance); helps us fine tune **openHandCoeff** and **closedHandCoeff**.

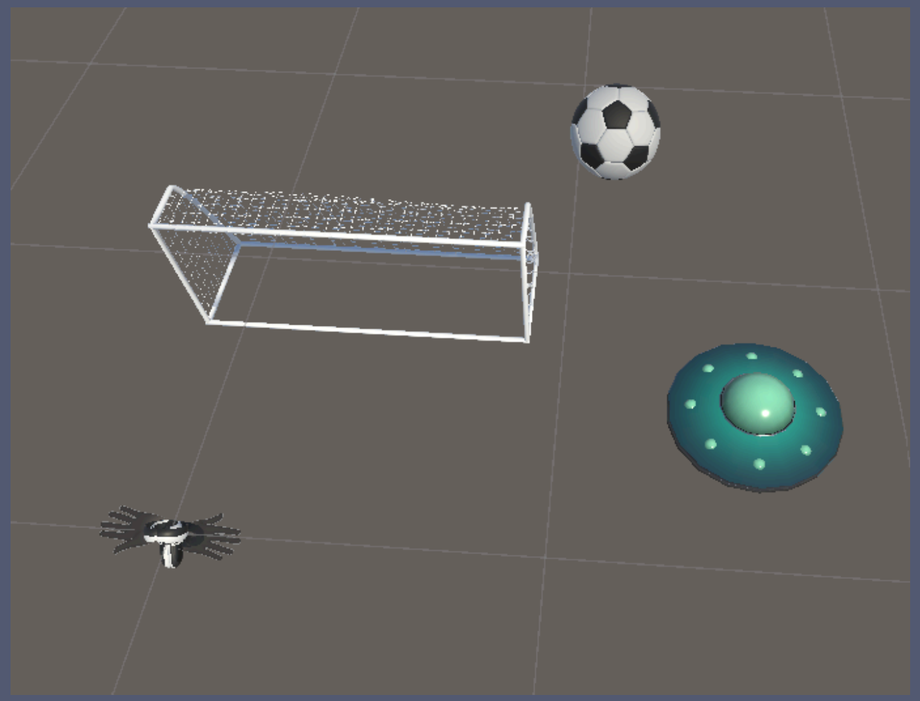
And we also modify **FixedUpdate()**

```
void FixedUpdate()
{
    Vector3 targetLocal = (handAnchor.position - eyeCenterAnchor.position).normalized;

    Vector3 dir = (puntaDito.position - eyeCenterAnchor.position).normalized;
    Vector3 targetLocal = dir * Mathf.Lerp(0f, maxDistanceUFO, currentOpennessCoeff);
    targetLocal += palmo.position;
    ...
}
```

As before we assign the right Transforms to the public fields of our script



And that's it! Our project is finished!



Thank you for the attention